SSCM Exercise 6

Nikolaus Czernin - 11721138

```
library("tidyverse")
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(knitr)
library("ISLR")
library("boot")

# Custom print function
print_ <- function(...) print(paste(...))

set.seed(11721138)
```

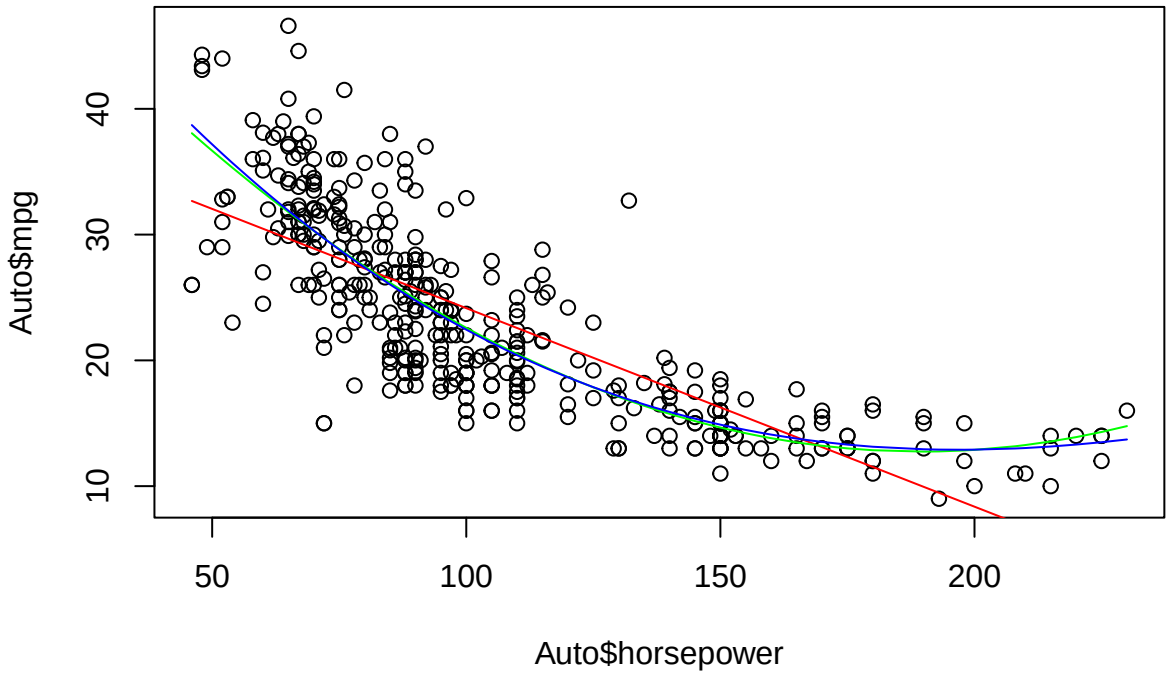
```
Auto %>% summary()
```

```
##       mpg          cylinders      displacement     horsepower        weight
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613
##  1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2225
##  Median :22.75   Median :4.000   Median :151.0   Median : 93.5   Median :2804
##  Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5   Mean   :2978
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0   3rd Qu.:3615
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140
##
##   acceleration        year           origin                      name
##  Min.   : 8.00   Min.   :70.00   Min.   :1.000   amc matador       :  5
##  1st Qu.:13.78   1st Qu.:73.00   1st Qu.:1.000   ford pinto        :  5
##  Median :15.50   Median :76.00   Median :1.000   toyota corolla    :  5
##  Mean   :15.54   Mean   :75.98   Mean   :1.577   amc gremlin       :  4
##  3rd Qu.:17.02   3rd Qu.:79.00   3rd Qu.:2.000   amc hornet        :  4
##  Max.   :24.80   Max.   :82.00   Max.   :3.000   chevrolet chevette:  4
##                                                  (Other)           :365
```

Task 1

Fitting basic models

```
# get the sorted indices of the predictor to allow for smooth line display later
idx.sorted <- order(Auto$horsepower)

# fit the models
model.1 <- lm(mpg ~ horsepower, data=Auto)
model.2 <- lm(mpg ~ poly(horsepower,2), data=Auto)
model.3 <- lm(mpg ~ poly(horsepower,3), data=Auto)

# plot the predictor against the response
plot(Auto$mpg ~ Auto$horsepower,)
lines(Auto$horsepower[idx.sorted], fitted(model.1)[idx.sorted], col='red')
lines(Auto$horsepower[idx.sorted], fitted(model.2)[idx.sorted], col='green')
lines(Auto$horsepower[idx.sorted], fitted(model.3)[idx.sorted], col='blue')
```



I use the lm() function to fix the 3 models. To plot them, I sort the points by the sorted indices of the horsepower variable.

The red line is the linear model, the green line is the quadratic model and the blue line is the cubic model. The polynomial models match curve-shape of the data nicely.

Validation set approach

Train-Test splitting

```
idx.train.50 <- sample(1:nrow(Auto), nrow(Auto)*0.5)
idx.train.70 <- sample(1:nrow(Auto), nrow(Auto)*0.7)
```

I generate a sample of the row indices, the size of 50% and 70% of the whole dataset.

```
train.50 <- Auto[idx.train.50,]
test.50 <- Auto[-idx.train.50,]
train.70 <- Auto[idx.train.70,]
test.70 <- Auto[-idx.train.70,]
```

To generate the train and test data sets, I use the generated indices.

```
# fit the models
model.1.50 <- lm(mpg ~ horsepower, data=train.50)
model.2.50 <- lm(mpg ~ poly(horsepower,2), data=train.50)
model.3.50 <- lm(mpg ~ poly(horsepower,3), data=train.50)
model.1.70 <- lm(mpg ~ horsepower, data=train.70)
model.2.70 <- lm(mpg ~ poly(horsepower,2), data=train.70)
model.3.70 <- lm(mpg ~ poly(horsepower,3), data=train.70)
```

```
get_rmse <- function(y, yhat) sqrt(mean((y - yhat)^2))
get_mse <- function(y, yhat) mean((y - yhat)^2)
get_mad <- function(y, yhat) median(abs(y - yhat))
```

These 3 function each take in the response and predicted response and return the wanted evaluation metrics.

Comparing the models

```
# make predictions for every model and report their performances
yhat.1.50 <- predict(model.1.50, test.50)
yhat.2.50 <- predict(model.2.50, test.50)
yhat.3.50 <- predict(model.3.50, test.50)
yhat.1.70 <- predict(model.1.70, test.70)
yhat.2.70 <- predict(model.2.70, test.70)
yhat.3.70 <- predict(model.3.70, test.70)

task1.results <- data.frame(
  Model=c("Linear 50/50","Quadratic 50/50","Cubic 50/50","Linear 70/30","Quadratic 70/30","Cubic 70/30"
  RMSE=c(get_rmse(test.50$mpg, yhat.1.50),
         get_rmse(test.50$mpg, yhat.2.50),
         get_rmse(test.50$mpg, yhat.3.50),
         get_rmse(test.70$mpg, yhat.1.70),
         get_rmse(test.70$mpg, yhat.2.70),
         get_rmse(test.70$mpg, yhat.3.70)
  ),
```

```
  MSE=c( get_mse(test.50$mpg, yhat.1.50),
         get_mse(test.50$mpg, yhat.2.50),
         get_mse(test.50$mpg, yhat.3.50),
         get_mse(test.70$mpg, yhat.1.70),
         get_mse(test.70$mpg, yhat.2.70),
         get_mse(test.70$mpg, yhat.3.70)
  ),
  MAD=c( get_mad(test.50$mpg, yhat.1.50),
         get_mad(test.50$mpg, yhat.2.50),
         get_mad(test.50$mpg, yhat.3.50),
         get_mad(test.70$mpg, yhat.1.70),
         get_mad(test.70$mpg, yhat.2.70),
         get_mad(test.70$mpg, yhat.3.70)
  )
)

task1.results %>% arrange(RMSE) %>% kable()
```

Model	RMSE	MSE	MAD
Quadratic 70/30	4.446434	19.77078	2.574765
Cubic 70/30	4.447284	19.77834	2.397192
Quadratic 50/50	4.648653	21.60998	2.460108
Cubic 50/50	4.655727	21.67580	2.430424
Linear 70/30	4.816677	23.20038	2.657085
Linear 50/50	5.285785	27.93952	3.087906

The 50/50-split linear model, and generally the linear models over the others, had the worst performances of all. The 70/30-split quadratic model performed best regarding all error measures, in second place came the cibic model with the same training split.

```
formeln <- list(
  "linear"=mpg ~ horsepower,
  "quadratic" = mpg ~ poly(horsepower, 2) ,
  "cubic" = mpg ~ poly(horsepower, 3)
)

# get cv error
get.cv.error <- function(formel, data, K=NULL){
  if (is.null(K)) K <- nrow(data)
  model <- glm(formel, data=data)
  cv = cv.glm(data, model)
  # the first value is the prediction error
  cv$delta[1]
}

task1.results <- task1.results %>% bind_cols(
  data.frame(
    LOO_error=c(
```

```
      # formeln %>% sapply(function(f) get.cv.error(f, Auto)),
      formeln %>% sapply(function(f) get.cv.error(f, Auto)) %>% rep(2)
    ),
    CV5_error=c(
      # formeln %>% sapply(function(f) get.cv.error(f, Auto, K=5)),
      formeln %>% sapply(function(f) get.cv.error(f, Auto, K=5))%>% rep(2)
    ),
    CV10_error=c(
      # formeln %>% sapply(function(f) get.cv.error(f, Auto, K=10)),
      formeln %>% sapply(function(f) get.cv.error(f, Auto, K=10))%>% rep(2)
    )
  )
)



task1.results %>% arrange(RMSE)  %>% kable()
```

Applying Cross-Validation

Model	RMSE	MSE	MAD	LOO_error	CV5_error	CV10_error
Quadratic 70/30	4.446434	19.77078	2.574765	19.24821	19.24821	19.24821
Cubic 70/30	4.447284	19.77834	2.397192	19.33498	19.33498	19.33498
Quadratic 50/50	4.648653	21.60998	2.460108	19.24821	19.24821	19.24821
Cubic 50/50	4.655727	21.67580	2.430424	19.33498	19.33498	19.33498
Linear 70/30	4.816677	23.20038	2.657085	24.23151	24.23151	24.23151
Linear 50/50	5.285785	27.93952	3.087906	24.23151	24.23151	24.23151

I iterate over the formulas this time around, as another coding workflow. I compute a generalized linear model and apply 1-fold (leave-one-out), 5-fold and 10-fold cross validation and extract the prediction error for all 6 models from before.

After performing cross validation, it seems like the 50/50-split models actually generalize a little better, as their prediction errors for any number of cross-correlation folds is lower than for the other models, even though their RMSE may not be the best of them in the first test.

I presume that in this case the models in the earlier tasks may have overfit to the fixed random split of the data. Cross correlation is meant to avoid this issue by testing the model multiple times to test k splits of the data, thus treating different subsets of the data as validtaion observations each time and getting closest to a real world scenario, where generalization would be most important.

In the first case, we provide no parameter K, which prompts the model to do leave-one-out cross correlation, which is essentially n-fold, where n is the number of observations in the data. It picks out a single observation from the data and makes only a prediction test on that single observation. In the second and third case, we make 5 and 10 equally-sized data-splits respectively.

Task 2

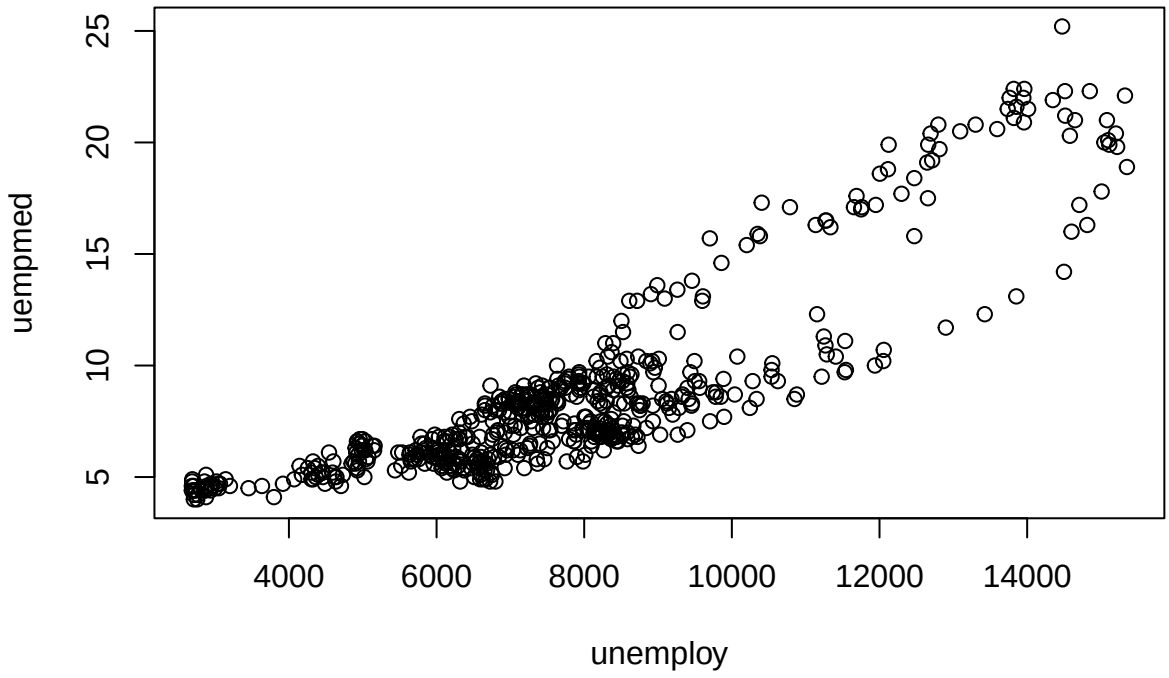
```
economics %>% summary()
```

```
##       date                 pce              pop            psavert
##  Min.   :1967-07-01   Min.   :  506.7   Min.   :198712   Min.   : 2.200
##  1st Qu.:1979-06-08   1st Qu.: 1578.3   1st Qu.:224896   1st Qu.: 6.400
##  Median :1991-05-16   Median : 3936.8   Median :253060   Median : 8.400
##  Mean   :1991-05-17   Mean   : 4820.1   Mean   :257160   Mean   : 8.567
##  3rd Qu.:2003-04-23   3rd Qu.: 7626.3   3rd Qu.:290291   3rd Qu.:11.100
##  Max.   :2015-04-01   Max.   :12193.8   Max.   :320402   Max.   :17.300
##     uempmed          unemploy
##  Min.   : 4.000   Min.   : 2685
##  1st Qu.: 6.000   1st Qu.: 6284
##  Median : 7.500   Median : 7494
##  Mean   : 8.609   Mean   : 7771
##  3rd Qu.: 9.100   3rd Qu.: 8686
##  Max.   :25.200   Max.   :15352
```

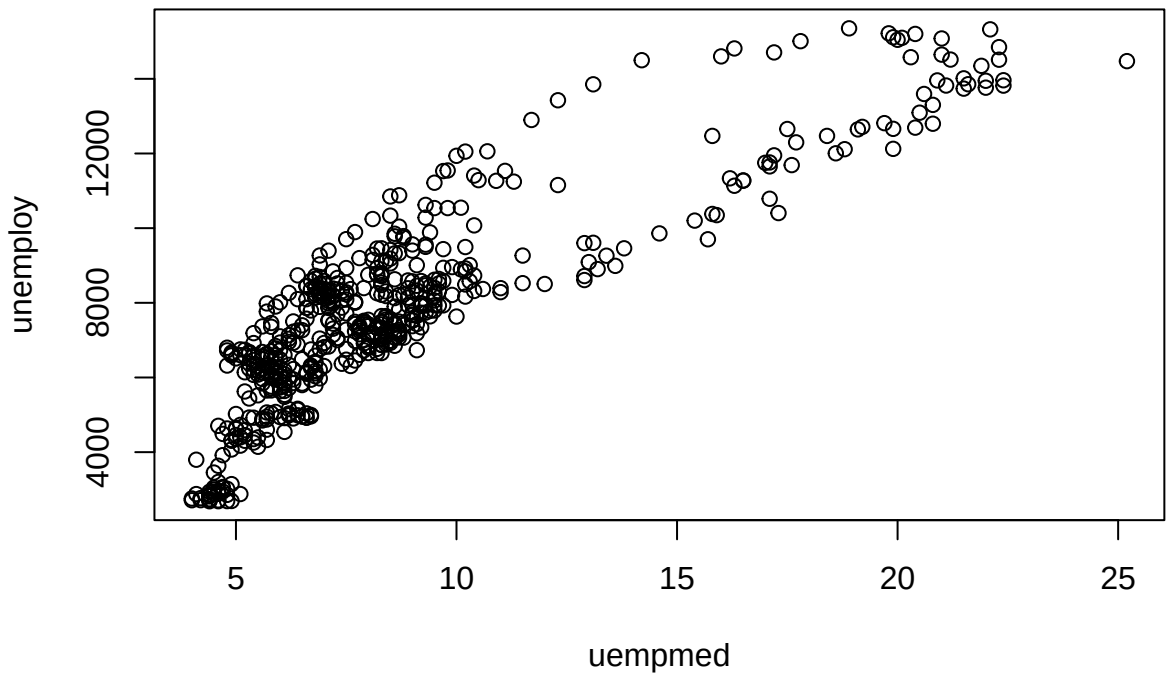
```
?economics
```

Basic models

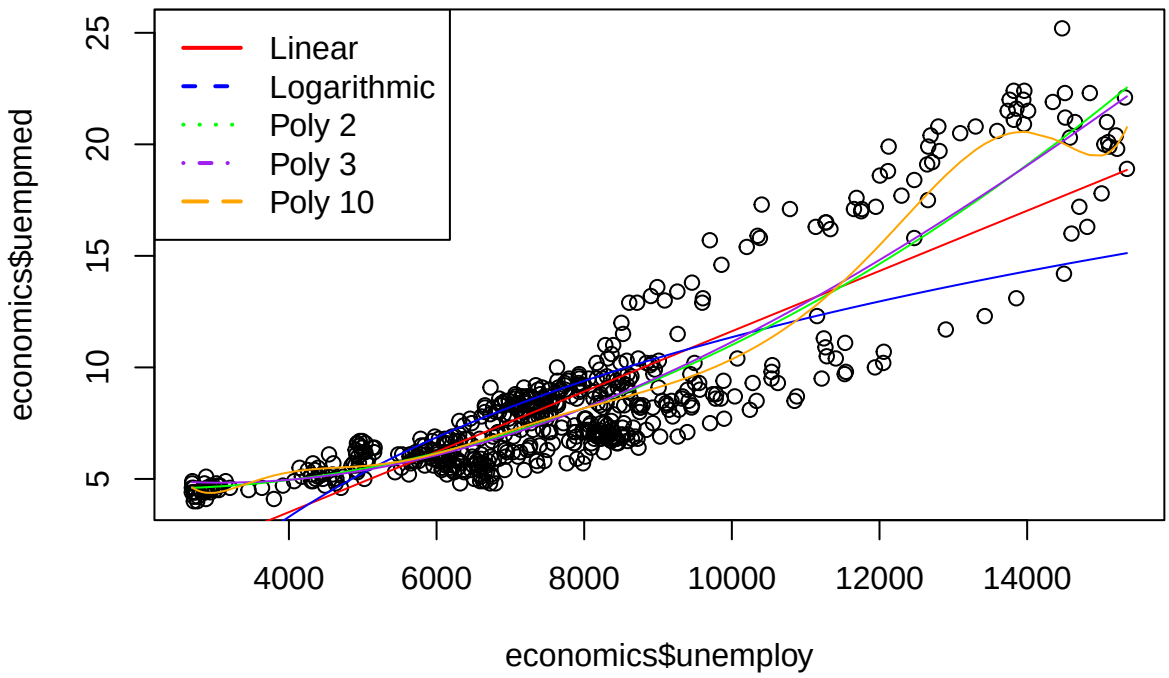
```
plot(uempmed ~ unemploy, data=economics)
```




```
plot(unemploy ~ uempmed, data=economics)
```

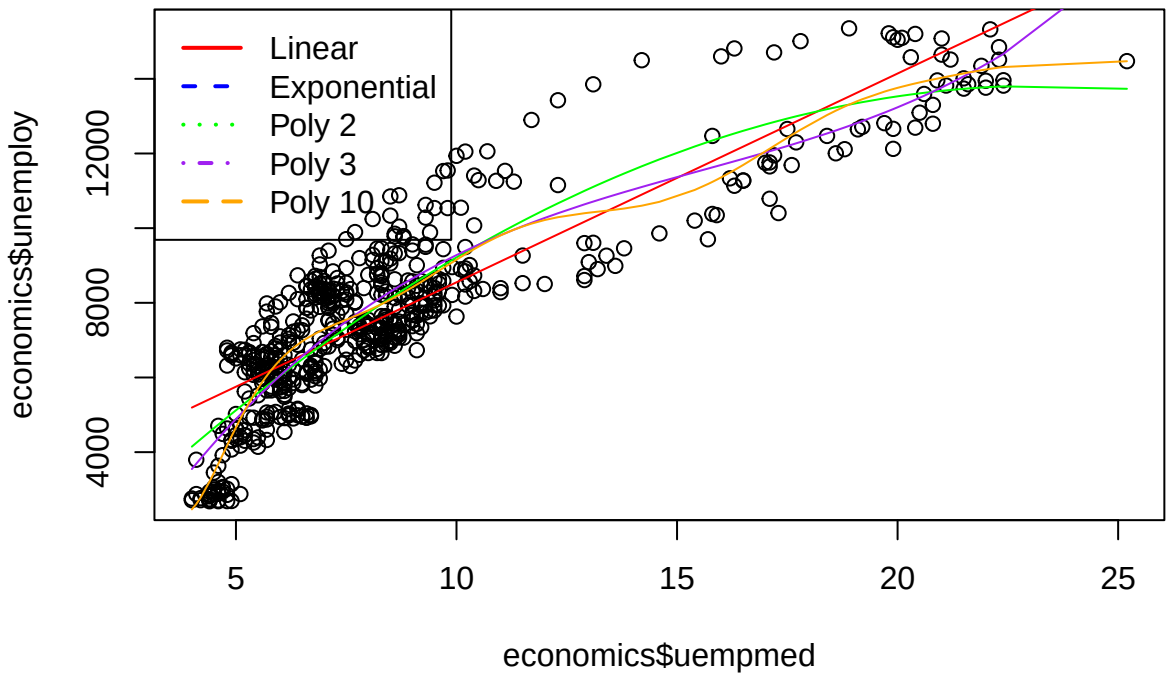



```
# sorted ids of unemploy for plotting
idx.daysbyemp.1 <- order(economics$unemploy)
# fitting the models
daysbyemp.1 <- lm(uempmed ~ unemploy, data=economics)
daysbyemp.log <- lm(uempmed ~ log(unemploy), data=economics)
daysbyemp.2 <- lm(uempmed ~ poly(unemploy, 2), data=economics)
daysbyemp.3 <- lm(uempmed ~ poly(unemploy, 3), data=economics)
daysbyemp.10 <- lm(uempmed ~ poly(unemploy, 10), data=economics)

# plot the predictor against the response
plot(economics$unemploy, economics$uempmed)
lines(economics$unemploy[idx.daysbyemp.1], fitted(daysbyemp.1)[idx.daysbyemp.1], col='red')
lines(economics$unemploy[idx.daysbyemp.1], fitted(daysbyemp.log)[idx.daysbyemp.1], col='blue')
lines(economics$unemploy[idx.daysbyemp.1], fitted(daysbyemp.2)[idx.daysbyemp.1], col='green')
lines(economics$unemploy[idx.daysbyemp.1], fitted(daysbyemp.3)[idx.daysbyemp.1], col='purple')
lines(economics$unemploy[idx.daysbyemp.1], fitted(daysbyemp.10)[idx.daysbyemp.1], col='orange')
# add a legend
legend("topleft", legend = c("Linear", "Logarithmic", "Poly 2", "Poly 3", "Poly 10"),
       col = c("red", "blue", "green", "purple", "orange"), lty = 1:5, lwd = 2)
```



Again, I had to sort the indices of the x-axis to display smooth lines. The "real" model of the data is not obvious because of the two line-patterns visible on the right end, which makes the decision of the "best" model difficult.

```
# sorted ids of unemploy for plotting
idx.daysbyemp.1 <- order(economics$uempmed)
# fitting the models
empbydays.1.1 <- lm(unemploy ~ uempmed, data=economics)
empbydays.1.log <- lm(log(unemploy) ~ uempmed, data=economics)
empbydays.1.2 <- lm(unemploy ~ poly(uempmed, 2), data=economics)
empbydays.1.3 <- lm(unemploy ~ poly(uempmed, 3), data=economics)
empbydays.1.10 <- lm(unemploy ~ poly(uempmed, 10), data=economics)

# plot the predictor against the response
plot(economics$uempmed, economics$unemploy)
lines(economics$uempmed[idx.daysbyemp.1], fitted(empbydays.1.1)[idx.daysbyemp.1], col='red')
lines(economics$uempmed[idx.daysbyemp.1], fitted(empbydays.1.log)[idx.daysbyemp.1], col='blue')
lines(economics$uempmed[idx.daysbyemp.1], fitted(empbydays.1.2)[idx.daysbyemp.1], col='green')
lines(economics$uempmed[idx.daysbyemp.1], fitted(empbydays.1.3)[idx.daysbyemp.1], col='purple')
lines(economics$uempmed[idx.daysbyemp.1], fitted(empbydays.1.10)[idx.daysbyemp.1], col='orange')
# add a legend
legend("topleft", legend = c("Linear", "Exponential", "Poly 2", "Poly 3", "Poly 10"),
       col = c("red", "blue", "green", "purple", "orange"), lty = 1:5, lwd = 2)
```



Performing cross-validation

```
# Predicting number of unemployed days by number of unemployed

# fitting the models
data.frame(
  model=c("linear", "logarithmic", "quadratic", "cubic", "^10"),
  LOO_mse = c(
    glm(uempmed ~ unemploy, data=economics) %>% cv.glm(economics, .) %>% .$delta %>% .[1],
    glm(uempmed ~ log(unemploy), data=economics) %>% cv.glm(economics, .) %>% .$delta %>% .[1],
    glm(uempmed ~ poly(unemploy, 2), data=economics) %>% cv.glm(economics, .) %>% .$delta %>% .[1],
    glm(uempmed ~ poly(unemploy, 3), data=economics) %>% cv.glm(economics, .) %>% .$delta %>% .[1],
    glm(uempmed ~ poly(unemploy, 10), data=economics) %>% cv.glm(economics, .) %>% .$delta %>% .[1]
  ),
  CV5_mse = c(
    glm(uempmed ~ unemploy, data=economics) %>% cv.glm(economics, ., K=5) %>% .$delta %>% .[1],
    glm(uempmed ~ log(unemploy), data=economics) %>% cv.glm(economics, ., K=5) %>% .$delta %>% .[1],
    glm(uempmed ~ poly(unemploy, 2), data=economics) %>% cv.glm(economics, ., K=5) %>% .$delta %>% .[1]
    glm(uempmed ~ poly(unemploy, 3), data=economics) %>% cv.glm(economics, ., K=5) %>% .$delta %>% .[1]
    glm(uempmed ~ poly(unemploy, 10), data=economics) %>% cv.glm(economics, ., K=5) %>% .$delta %>% .[1]
  ),
  CV10_mse = c(
    glm(uempmed ~ unemploy, data=economics) %>% cv.glm(economics, ., K=10) %>% .$delta %>% .[1],
    glm(uempmed ~ log(unemploy), data=economics) %>% cv.glm(economics, ., K=10) %>% .$delta %>% .[1],
    glm(uempmed ~ poly(unemploy, 2), data=economics) %>% cv.glm(economics, ., K=10) %>% .$delta %>% .[1]
    glm(uempmed ~ poly(unemploy, 3), data=economics) %>% cv.glm(economics, ., K=10) %>% .$delta %>% .[1]
    glm(uempmed ~ poly(unemploy, 10), data=economics) %>% cv.glm(economics, ., K=10) %>% .$delta %>% .[1]
```

```
  )
) %>%
  mutate(
    LOO_rmse=sqrt(LOO_mse),
    CV5_rmse=sqrt(CV5_mse),
    CV10_rmse=sqrt(CV10_mse),
    ) %>%
  kable()
```

model	LOO_mse	CV5_mse	CV10_mse	LOO_rmse	CV5_rmse	CV10_rmse
linear	4.159797	4.160685	4.151073	2.039558	2.039776	2.037418
logarithmic	6.998858	6.964840	7.045579	2.645536	2.639098	2.654351
quadratic	3.005112	3.028514	2.996565	1.733526	1.740263	1.731059
cubic	3.009934	2.989106	3.033975	1.734916	1.728903	1.741831
^10	2.832303	2.912133	2.836316	1.682945	1.706497	1.684137

The first results show that the linear model underfits the data with high errors, while the logarithmic model performs even worse, showing it is not suitable. The quadratic and cubic models perform similarly, with the quadratic model slightly better. The 10th-degree polynomial has the lowest errors but risks overfitting due to its high complexity.

```
# Predicting number of unemployed days by number of unemployed

# fitting the models
data.frame(
  model=c("linear", "logarithmic", "quadratic", "cubic", "^10"),
  LOO_mse = c(
    glm(unemploy ~ uempmed, data=economics) %>% cv.glm(economics, .) %>% .$delta %>% .[1],
    glm(log(unemploy) ~ uempmed, data=economics) %>% cv.glm(economics, .) %>% .$delta %>% .[1],
    glm(unemploy ~ poly(uempmed, 2), data=economics) %>% cv.glm(economics, .) %>% .$delta %>% .[1],
    glm(unemploy ~ poly(uempmed, 3), data=economics) %>% cv.glm(economics, .) %>% .$delta %>% .[1],
    glm(unemploy ~ poly(uempmed, 10), data=economics) %>% cv.glm(economics, .) %>% .$delta %>% .[1]
  ),
  CV5_mse = c(
    glm(unemploy ~ uempmed, data=economics) %>% cv.glm(economics, ., K=5) %>% .$delta %>% .[1],
    glm(log(unemploy) ~ uempmed, data=economics) %>% cv.glm(economics, ., K=5) %>% .$delta %>% .[1],
    glm(unemploy ~ poly(uempmed, 2), data=economics) %>% cv.glm(economics, ., K=5) %>% .$delta %>% .[1]
    glm(unemploy ~ poly(uempmed, 3), data=economics) %>% cv.glm(economics, ., K=5) %>% .$delta %>% .[1]
    glm(unemploy ~ poly(uempmed, 10), data=economics) %>% cv.glm(economics, ., K=5) %>% .$delta %>% .[1]
  ),
  CV10_mse = c(
    glm(unemploy ~ uempmed, data=economics) %>% cv.glm(economics, ., K=10) %>% .$delta %>% .[1],
    glm(log(unemploy) ~ uempmed, data=economics) %>% cv.glm(economics, ., K=10) %>% .$delta %>% .[1],
    glm(unemploy ~ poly(uempmed, 2), data=economics) %>% cv.glm(economics, ., K=10) %>% .$delta %>% .[1]
    glm(unemploy ~ poly(uempmed, 3), data=economics) %>% cv.glm(economics, ., K=10) %>% .$delta %>% .[1]
    glm(unemploy ~ poly(uempmed, 10), data=economics) %>% cv.glm(economics, ., K=10) %>% .$delta %>% .[1]
  )
) %>%
  mutate(
    LOO_rmse=sqrt(LOO_mse),
    CV5_rmse=sqrt(CV5_mse),
```

```
    CV10_rmse=sqrt(CV10_mse),
    ) %>%
  kable()
```

model	LOO_mse	CV5_mse	CV10_mse	LOO_rmse	CV5_rmse	CV10_rmse
linear	1.715211e+06	1.708361e+06	1.715714e+06	1309.660569	1307.0427440	1309.8528461
logarithmic	5.311920e-02	5.337120e-02	5.323230e-02	0.230476	0.2310221	0.2307212
quadratic	1.432531e+06	1.428169e+06	1.433665e+06	1196.883710	1195.0600742	1197.3574129
cubic	1.366405e+06	1.361970e+06	1.364481e+06	1168.933099	1167.0348667	1168.1098749
^10	4.530738e+06	5.766625e+06	1.570784e+06	2128.553095	2401.3798824	1253.3092416

The logarithmic model performs exceptionally well with very low errors, while all other models, including the quadratic, cubic, and 10th-degree, have much higher errors. This suggests that the logarithmic relationship is more appropriate for this direction.

Cross-validation helps identify underfitting (simple models like linear) and overfitting (complex models like the 10th-degree polynomial). Leave-one-out cross validation (LOO) is very precise but computationally expensive, because it computes results for each row of the dataset. while 5-fold and 10-fold CV are faster and still effective. The best model generally should balance errors across all CV methods, showing both good fit and generalizability.